

# Structuring Agents for Adaptation

Sander van Splunter, Niek J.E. Wijngaards, and Frances M.T. Brazier

Intelligent Interactive Distributed Systems Group, Faculty of Sciences, Vrije Universiteit  
Amsterdam, de Boelelaan 1081a, 1081HV, The Netherlands  
{sander,niek,frances}@cs.vu.nl  
<http://www.iids.org/>

**Abstract.** Agents need to be able to adapt to the dynamic nature of the environments in which they operate. Automated adaptation is an option that is only feasible if enough structure is provided. This paper describes a component-based structure within which dependencies between components are made explicit. An example of a simple web-page analysis agent is used to illustrate the structuring principles and elements.

## 1 Introduction

Agents typically operate in dynamic environments. Agents come and go, objects and services appear and disappear, and cultures and conventions change. Whenever an environment of an agent changes to the extent that an agent is unable to cope with (parts of) the environment, an agent needs to adapt. Changes in the social environment of an agent, for example, may require new agent communication languages, or new protocols for auctions. In some cases an agent may be able to detect gaps in its abilities; but not be able to fill these gaps on its own (with, e.g., its own built-in learning and adaptation mechanisms).

Adaptive agents are a current focus of research (e.g., see this book), but opinions on what 'adaptation' constitutes differ. Sometimes reactive behaviour of an agent is dubbed 'adaptive behaviour' [1] where an agent is, e.g., capable of abandoning a previous goal or plan and adopting a new goal or plan that fits its current situation better. In this paper, adaptation of an agent is used to refer to "structural" changes of an agent, including knowledge and facts available to an agent. External assistance may be needed to perform the necessary modifications, e.g. by an agent factory [2].

An adaptation process has a scope: a scope defines the extent to which parts of an agent are adapted. Research on agent adaptation can be categorised by distinguishing three specific scopes: adaptation of knowledge and facts; adaptation of the language with which an agent's interface to the outside world is expressed (e.g., dependency on agent platform), and adaptation of an agent's functionality.

- Research on *adaptation of knowledge and facts* of an agent is usually based on (machine) learning, e.g. [3]. Example applications include personification: an agent maintains and adapts a profile of its (human) clients, e.g. [4], [5] and [6], co-

- ordination in multi-agent systems, e.g. [7] and [8], and situated learning for agents, e.g. [9].
- Research on *adaptation of the interface* of an agent is usually concerned with adapting the agent's interface to the (current) agent platform, e.g. see [10], [11].
- Research on *adapting an agent's functionality* is not commonly available. Agent creation tools are usually semi-automatic, providing a basis for developing an automated tool for agent adaptation, e.g. see AGENTBUILDER [12], D'AGENTS/AGENTTCL [13], ZEUS [14], and PARADE [15]. Computer assisted software engineering tools are usually not focussed on agents, and are less concerned with 'adaptivity'; see the discussion in Section 4 for a more detailed comparison.

The approach taken in this paper focuses on automated adaptation of an agent's functionality by means of an agent factory. An agent factory is an external service that adapts agents, on the basis of a well-structured description of the software agent. Our hypothesis is that structuring an agent makes it possible to reason about an agent's functionality on the basis of its blueprint (that includes information about its configuration). This ability makes it possible to identify specific needs for change, defining the necessary input required to automatically reconfigure an agent. This approach is much akin to the knowledge-level approach to system design [16] in which the knowledge-level is distinguished from the symbol level. The agent factory presented in this paper relies on a component-based agent architecture described in Section 2. An example of the use of these component-based structures by (automated) adaptation of a simple web-page analysis agent is shown in Section 3. Section 4 discusses results of this approach.

## 2 Structure of Agents

The structure of an agent proposed in this paper is based on general software engineering, knowledge engineering and agent technology principles. Section 2.1 briefly discusses these principles. Section 2.2 describes the structuring principles used in this paper. The result is illustrated for a simple web analysis agent introduced in Section 2.3.

### 2.1 Structuring Principles

Intelligent agents are autonomous software programs that exhibit social, co-operative, and intelligent behaviour in distributed environments [17]. Modelling and implementing 'intelligent' agents are not only studied in Software Engineering, but also in Knowledge Engineering and Agent Research. Each of these research disciplines imposes its own structuring principles, often adopting principles of other research disciplines.

In *software engineering* functional programming, e.g. [18], object-oriented programming, e.g. [19], and component-based programming, e.g. [20], [21] have a

concept of compositionality. The compositional structure in functional programming and component-based programming is based on processes and functions. The concept of compositionality in object oriented programming is that of objects that encapsulate data and methods. Each approach has its own merits, depending on characteristics of the domain for which a software system is designed. Re-use and maintenance are important aspects of all approaches (see e.g. [22]). All require a means to specify and retrieve appropriate software components [23]: by means of e.g. design patterns [24], annotation of software components [25], and annotation of web-services [26].

In *knowledge engineering*, common structuring principles have a process-oriented focus, in which the (problem solving / reasoning) processes are explicitly modelled and controlled, e.g. by approaches related to COMMONKADS [27] and DESIRE [28]. Methodologies including reasoning patterns and generic task models have been defined, facilitating re-use and maintenance.

In *intelligent agent research* a wide variety of approaches are employed. Most common seem to be process (or task) oriented approaches, for which general agent models are defined, e.g. by INTERRAP [29], ZEUS [14], and DESIRE [28]. An example of a common model is the BDI architecture, proposed by [30].

Each of the approaches described above employs a notion of compositionality and defines generic models or patterns. Reuse and maintenance are recognised as important endeavours, but as such, not often formalised nor automated. Current research on brokering for web-services focuses on annotations of services (roughly stated: software components). Annotations make architectural assumptions explicit, including assumptions about the nature of the components, the nature of the connectors, the global architectural structure, and the construction process [31].

## 2.2 Agent Structuring Approach

For automated agent adaptation, an agent structuring approach is needed which facilitates reuse of existing components of an agent. This implies explication of not only the structure of reusable parts, but also the semantics, including assumptions and behaviour.

The component-based approach proposed in this paper distinguishes *components* and *data types* (akin to data formats) [32] incorporating process-oriented and object-oriented approaches. Where process-oriented modelling approaches distinguish processes and information exchange explicitly, object-oriented approaches encapsulate data and methods in objects. In the approach proposed in this paper components are the 'active' parts of an agent (akin to processes), and data types are the 'passive' parts of an agent (akin to classes). This approach combines process-oriented and object-oriented approaches, building on knowledge-engineering and software-engineering research results.

Components have an interface, describing their input and output data types, and *slots* within which component configuration is regulated. *Data types* represent types of data, and may have their own slots with which data type configuration is regulated. Slots define their interface and that which is expected from the component or data type

that is to be inserted. The addition of slots makes components not "black boxes", but "grey boxes"; components can thus be partial specifications. De Bruijn and van Vliet [33] even argue that for reuse a "black box" approach to components in component-based development is a dead end. The concept of slots helps defining the 'static' structure or architecture of an agent. Components and data types need to be matched to slots, determining, as a result, matches between, e.g., (replaceable) software components [34].

*Co-ordination patterns* and *ontologies* are distinguished to annotate configurations of components and data types. Annotations are essential for automation of the agent adaptation process. A co-ordination pattern is a temporal specification (e.g., see [28]) defining temporal relations and dependencies between processes, when used within a specific task. A co-ordination pattern describes the flow of control and information for group of components in the context of a specific task. An ontology describes the concepts and relations between concepts. Co-ordination patterns and ontologies may themselves be composed and are ultimately related to (annotations of) components and data types.

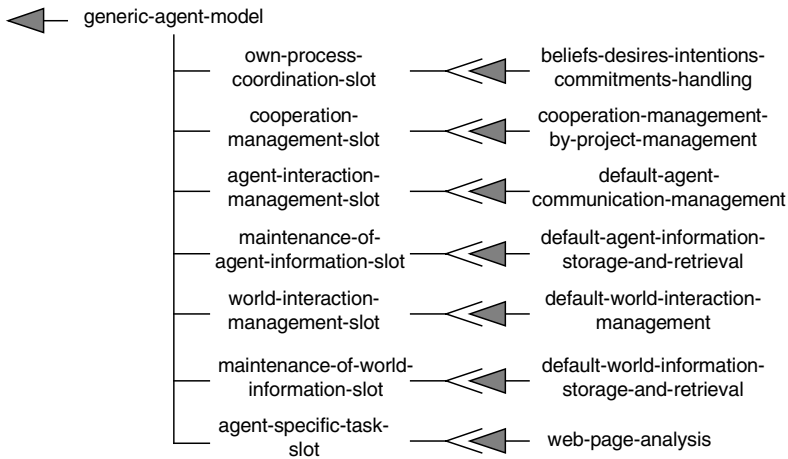
## 2.3 An Example

To illustrate the role of structure in our approach a web-analyser agent is introduced; an agent that analyses websites for relevance, on demand. Given a URL of a website and a term, a web analyser agent determines the relevance of the website with regard to the given term. The agent uses simple analysis techniques: it counts the number of occurrences of the term on the pages at the given location. Three components of the agent are described to illustrate the agent's functionality and component configuration.

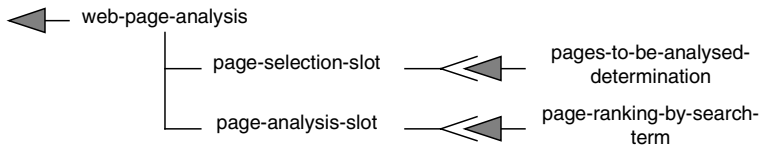
The web-analyser agent's major structuring component is the *generic-agent-model* component [28]. The *generic-agent-model* component models an agent that can reason about its own processes, interact with and maintain information about other agents, interact with and maintain information about the external world, co-operate with other agents, and performs its own specific tasks. Figure 1 shows the compositional structure of the *generic-agent-model* component and its seven component slots. For each slot, the name of the component inserted into the slot is given.

The further structure of the *conceptual* component *web-page-analysis* inserted in the *agent-specific-task-slot* of the *conceptual generic-agent-model* component is shown in Figure 2.

The generic agent model can be used to structure both the conceptual and operational description of an agent. At *operational* level the components within the *web-page-analysis* component differ from the components in the conceptual description, as shown in Figure 3. The *conceptual page-ranking-by-search-term* component is implemented by the operational component configuration of the operational *two-set-enumerator* and *count-substring-in-string* components.

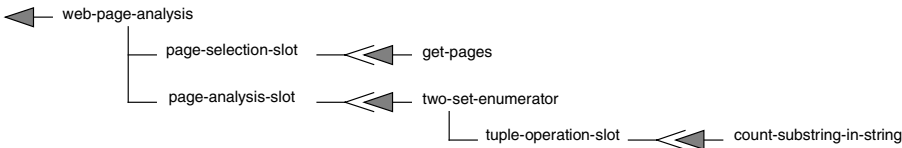


**Fig. 1.** The generic-agent model structure for the simple web analyser agent at conceptual level.



**Fig. 2.** The structure of the web-page-analysis component at conceptual level.

A rationale for this operational configuration is that a set of webpages needs to be ranked for one search term. The actual analysis process consists of counting the number of occurrences of the search term in a web page by simply counting the number of occurrences of a (small) string in another (larger) string, i.e. a web page.



**Fig. 3.** Structure of the web-page-analysis component at operational level.

Co-ordination patterns are used to verify whether the configuration of components and data types exhibits the required behaviour, in this case receiving requests for web analysis, performing requested web analysis, and returning results. A high-level co-ordination pattern for multiple job execution is applicable; a "job" is a "request for web analysis". In this specific case, a simple sequential job execution pattern suffices. This co-ordination pattern is shown in pseudo-code below: "tasks" are ordered in time, and need to be performed by the configuration proposed.

- (1) collect jobs in job list
- (2) select a job
- (3) perform current selected job
- (4) remove current selected job from job lists
- (5) go to (1)

The tasks shown in the co-ordination pattern may be directly mapped onto components, but this is not necessarily the case. Some of the tasks may involve a number of components. For example, the first task, collect jobs in job list involves, from the perspective of the generic-agent-model component, components in two of its 'slots': agent-interaction-management for receiving web-analysis requests, and maintenance-of-agent-information for storing web-analysis requests. Another co-ordination pattern, collect items in existing item list, is needed to specify this task more precisely. These tasks can be mapped directly onto the above mentioned components.

- (1a) obtain item
- (1b) add obtained items to item list

### 3 Adapting Structured Agents

This section describes how agents with a compositional structure as described in the previous section can be adapted. Section 3.1 introduces the adaptation process of the agent factory, and Section 3.2 describes the results of adapting a simple web-page analysis agent.

#### 3.1 Adaptation in an Agent Factory

Agents are constructed from components and data types by an automated agent factory [2]. Adapting an agent entails adapting the configuration of its components and data types. Whether the need for servicing is detected by an agent itself, or by another agent (automated or human) is irrelevant in the context of this paper. The agent factory is based on three underlying assumptions: (1) agents have a compositional structure with reusable parts, (2) two levels of conceptualisation are used: conceptual and operational, (3) re-usable parts can be annotated and knowledge about annotations is available.

An agent factory, capable of automatically building and adapting an agent, combines knowledge of its domain (i.e., adapting *intelligent agents*), its process (i.e., *adaptation* processes), and the combination of these (i.e., *adapting intelligent agents*). Needs for adaptation are qualified to express preference relations among needs, and refer to properties of an agent. Needs may change during the adaptation process, e.g. conflicting needs may be removed.

An adaptation process starts with a blueprint of an agent, and results in a (modified) blueprint of the agent; a process similar to re-design processes. In re-design, an initial artefact is modified according to new, additional, requirements. An existing model of

re-design [35] has been used to model the adaptation process. Models and systems for re-design make use of the structure of their artefacts, the same holds for the adaptation process. Strategic knowledge is required to 'guide' the adaptation process, both in deciding which goals to pursue and how to tackle a goal; goals take the form of adaptation foci. An adaptation focus consists of the following categories of elements of the agent:

- needs that are taken into account: e.g. needs that relate to a specific facet (task or function, data, behaviour) of part of the agent,
- levels of conceptualisation
- components
- data types
- co-ordination patterns and their mappings
- ontologies and their mappings
- annotations

The component-based adaptation approach presented in this paper is similar to *design-as-configuration*, e.g., as described in [8], which focuses on constructing a satisfactory configuration of elements on the basis of a given set of requirements (also named: constraints). Examples of design-as-configuration are described in COMMON-KADS [27] and an analysis of elevator configuration systems [36].

### 3.2 Adaptation Results

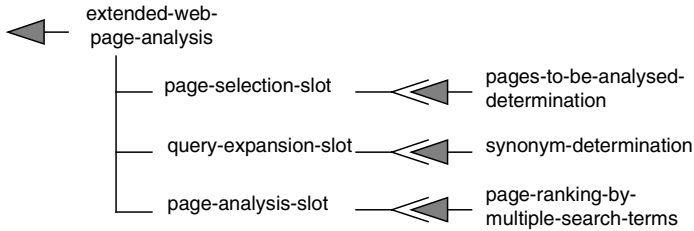
Assume in the example introduced in Section 2.2 that the owner of the web analyser agent has decided that she wants to be able to acquire a higher level of service for those sites for which she is known to be a preferred client (and the standard quality of service for those sites for which this is not the case). The (new) requirements for the web-analyser agent are that:

- the agent shall have two levels of quality of service for assessing relevance of web pages;
- the agent shall employ other analysis methods in addition to its analysis based on a single search term: analysis involving synonyms is a better quality of service than analysis involving a single search term;
- the agent shall maintain a list of those sites for which its client is a preferred client;
- the agent shall be informed about a site's preferred clients;
- a co-ordination pattern shall relate a client's request to a preferred quality of service.

The resulting blueprint is described in this section by focusing on the changes within the conceptual agent-specific-task component, the most constrained component of the agent. Other components and data types are not shown in this description.

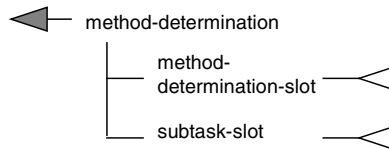
In one of the libraries of components and data types, an alternative web-page-analysis component is found which has a slot for query expansion, shown in Figure 4. The alternative quality of service for web-page analysis consists of expanding a search term into a set of synonyms with which web-pages are analysed. The slots of this component can be filled with components used in the original web-page-analysis

component. One new component needs to be found, to ‘expand a search term. A component that uses a synonym database qualifies, and is used.



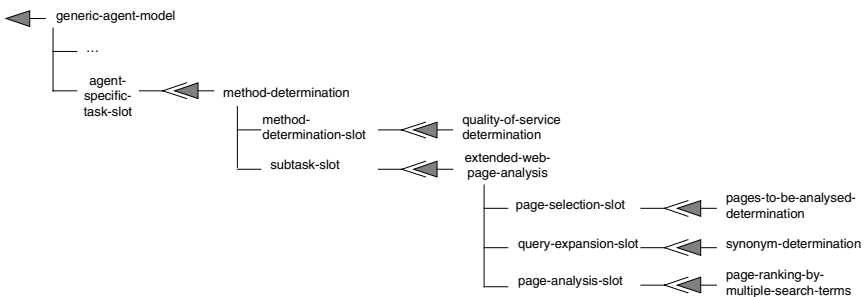
**Fig. 4.** Component extended-web-page-analysis at conceptual level.

This extended-web-page-analysis component is parameterised, i.e., the level of query expansion can be specified explicitly. This property makes it possible to provide both required qualities of service: one quality of service with extended query expansion, the other quality of service with no query expansion at all. The existing web-page-analysis component can be replaced by the extended-web-page-analysis component. An additional component is needed to determine the applicable quality of service, as shown in Figure 5.



**Fig. 5.** Component method deliberation at conceptual level.

The resulting component configuration within agent-specific-task is shown in Figure 6.



**Fig. 6.** The agent-specific-task-slot contains a conceptual component for selecting a quality of service of web-page analysis.

The same high-level co-ordination pattern is used as described in Section 2.2, however the third task, perform current selected job, has been replaced by a different (sub)co-



ordination pattern which involves the choice of a specific quality of service. This co-ordination pattern is shown below:

- (3a) prepare for current job
- (3b) plan work for current job
- (3c) perform planned work for current job
- (3d) finish current job

The main change is in the presence of the second sub-task, plan work for current job, which is related to the `method_determination` component. The changes in the operational configuration of components and data types for the resulting agent, are comparable to those needed for the conceptual configuration of components and data types.

## 4 Discussion

Agents can be adapted by services such as an agent factory. Automated adaptation of software agents is a configuration-based process requiring explicit knowledge of strategies to define and manipulate adaptation foci. Automated agent adaptation becomes feasible when the artefact is structured, as demonstrated in a number of prototypes. A compositional approach is taken to structure the agent: components and data-types can be configured to form an agent, together with co-ordination patterns and ontologies which describe the agent's behaviour and semantics. A simple web-page analysis agent has been used to illustrate the agent structuring and adaptation process needed to adapt an agent's functionality.

An example of the use of an agent factory for the adaptation of the external interface of an agent (a less complex endeavour) is described in [11]. For agents that migrate in an open, heterogeneous environment generative migration minimally entails adapting an agent's wrapper. It may, however, also involve rebuilding an agent with different operational components and data types (e.g., in a different code base). Four different scenario's for generative migration have been distinguished: homogeneous, cross-platform, agent-regeneration, and heterogeneous migration. Migration is categorised with respect to combinations of variation of (virtual) machines and agent platforms.

The proposed structuring of agents presented in this paper is similar to other work, in the eighties, in which an automated software design assistant is developed [37]. To facilitate automated derivation of a structural design from a logical model of a system, a modular structure was assumed, with the explicit property that independent modules are clearly understood, together with explicit, dependencies between modules. In their approach, a logical description of processes is modularised. This technique has shown to be useful, on the one hand, for grouping functionality and tasks into components and co-ordination patterns, and on the other hand for grouping needs for adaptation.

The Programmer's Apprentice [38], from the same period, aims to provide intelligent assistance in all phases of the programming task: an interactive tool that may alleviate a programmer of routine tasks. By using 'clichés', patterns of code, the system can 'understand' parts of code. This work is not based on components, but on

individual programming statements, which is a major difference with our work. A number of the processes involved in the Programmer's Apprentice are of relevance to the adaptation process. A related semi-automated approach, KIDS [39], derives programs from formal specifications. In this approach users interactively transform a formal specification into an implementation; this is mainly used for algorithm design. The principles apply to our approach for, e.g., adapting an operational configuration of components and data types on the basis of an already adapted conceptual configuration of components and data types.

The adaptation approach taken in this paper is similar to approaches such as IBROW [40]. IBROW supports semi-automatic configuration of intelligent problem solvers. Their building blocks are 'reusable components', which are not statically configured, but dynamically 'linked' together by modelling each building block as a CORBA object. The CORBA-object provides a wrapper for the actual implementation of a reusable component. A Unified Problem-solving method development language UPML [41] has been proposed for the conceptual modelling of the building blocks. Our approach differs in a number of aspects, which include: no commitments to specific conceptual or operational languages and frameworks, two levels of conceptualisation, and the process of (re-)configuration is a completely automated (re-)design process.

The design of an agent within the agent factory is based on configuration of components and data types. Components and data types may include cases and partial (agent) designs (cf. generic models / design patterns). This approach is related to design patterns (e.g., [24], [42], [43]) and libraries of software with specific functionality (e.g., problem-solving models, e.g. [27], or generic task models, e.g. [28]). The adaptation process uses strategic knowledge to explore the design space of possible configurations with the aim of satisfying the needs for adaptation. Alternative approaches may expand *all* configurations of (some) components and data types, when insufficient knowledge on their (non)functional properties is available [44].

Module Interconnection Languages [45] explicitly structure interfaces and relations between components, providing a basis for matching descriptions of components and slots [34]. Approaches for semantic web and annotation of web services play an important role in representing and reasoning about annotations [26].

The QUASAR project focuses on (semi-)automation of generation and provision of implementations of software architectures. A reference architecture is derived based on functional requirements, after which modifications are made for non-functional requirements. Their approach is based on top-down software architecture decomposition, where choices are based on Feature-Solution graphs, which link requirements to design solutions [46], [47].

Future research focuses on augmenting our prototype and analysing annotations in the context of semantic web research.

**Acknowledgements.** The authors wish to thank the graduate students Hidde Boonstra, David Mobach, Oscar Scholten and Mark van Assem for their explorative work on the application of an agent factory for an information retrieving agent. The authors are also grateful to the support provided by Stichting NLnet, <http://www.nlnet.nl/>.

## References

1. Rus, D., Gray, R.S., Kotz, D.: Autonomous and Adaptive Agents that Gather Information. In: Proceedings of AAAI'96 International Workshop on Intelligent Adaptive Agents (1996) 107–116
2. Brazier, F.M.T., Wijngaards, N.J.E.: Automated Servicing of Agents. AISB Journal **1** (1), Special Issue on Agent Technology (2001) 5–20
3. Kudenko, D., Kazakov, D., Alonso, E.: Machine Learning for Multi-Agent Systems. In: V. Plekhanova, V.(ed.): Intelligent Agents Software Engineering, Idea Group Publishing (2002)
4. Bui, H.H., Kieronska, D., and Venkatesh, S.: Learning Other Agents' Preferences in Multiagent Negotiation. In: Proceedings of the National Conference on Artificial Intelligence (AAAI-96) (1996) 114–119
5. Soltysiak, S., Crabtree, B.: Knowing me, knowing you: Practical issues in the personalisation of agent technology. In: Proceedings of the third international conference on the practical applications of intelligent agents and multi-agent technology (PAAM98), London (1998) 467–484
6. Wells, N., Wolfers, J.: Finance with a personalized touch. Communications of the ACM, Special Issue on Personalization **43**:8 (2000) 31–34.
7. Schaerf, A., Shohamm Y., Tennenholtz, M.: Adaptive Load Balancing: A Study in Multi-Agent Learning. Journal of Artificial Intelligence Research **2** (1995) 475–500
8. Stefik, M.: Introduction to Knowledge Systems. Morgan Kaufmann Publishers, San Francisco, California (1995)
9. Reffat, R.M. and Gero, J.S.: Computational Situated Learning in Design. In: J. S. Gero (ed.): Artificial Intelligence in Design '00. Kluwer Academic Publishers, Dordrecht (2000) 589–610
10. Brandt, R., Hörtnagl, C., Reiser, H.: Dynamically Adaptable Mobile Agents for Scaleable Software and Service Management. Journal of Communications and Networks **3**:4 (2001) 307–316
11. Brazier, F.M.T., Overeinder, B.J., van Steen, M., Wijngaards, N.J.E.: Agent Factory: Generative Migration of Mobile Agents in Heterogeneous Environments. In: Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002) (2002) 101–106
12. Reticular: AgentBuilder: An Integrated Toolkit for Constructing Intelligent Software Agents. Reticular Systems Inc, white paper edition. <http://www.agentbuilder.com> (1999)
13. Gray, R.S., Kotz, D., Cybenko, G., Rus, D.: Agent Tcl. In: Cockayne, W., Zypa, M. (eds.): Itinerant Agents: Explanations and Examples with CD-ROM. Manning Publishing. (1997) 58–95
14. Nwana, H.S., Ndumu, D.T., Lee, L.C.: ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems. Applied AI **13**:1/2 (1998) 129–185
15. Bergenti, F., Poggi A.: A Development Toolkit to Realize Autonomous and Inter-Operable Agents. In: Proceedings of Fifth International Conference of Autonomous Agents (Agents 2001), Montreal (2001) 632–639

16. Newell, A.: The Knowledge Level. *Artificial Intelligence* **18**:1 (1982) 87–127.
17. Jennings, N.R., Wooldridge, M.J.: Applications Of Intelligent Agents. In: Jennings, N.R., Wooldridge, M.J. (eds.): *Agent Technology Foundations, Applications, and Markets*, Springer-Verlag, Heidelberg, Germany (1998) 3–28
18. Kernighan, B.W., Ritchie, D.M.: *The C Programming Language*. 2nd edn. Prentice Hall Software Series (1988)
19. Booch, G.: *Object oriented design with applications*. Benjamins Cummins Publishing Company, Redwood City (1991)
20. Hopkins, J.: Component primer. *Communications of the ACM* **43**:10 (2000) 27–30
21. Sparling, M.: Lessons learned through six years of component-based development. *Communications of the ACM* **43**:10 (2000) 47–53
22. Biggerstaff, T., Perlis, A. (eds.): *Software Reusability: Concepts and models*. Vol 1. New York, ACM Press (1997)
23. Moormann Zaremski, A., Wing, J.M.: Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 6:4 (1997) 333–369
24. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of reusable object-oriented software*. Addison Wesley Longman, Reading, Massachusetts (1994)
25. Shaw, M., DeLine, R., Klein, D.V., Ross, T.L., Young, D.M., Zelesnik G.: Abstractions for Software Architecture and Tools to Support Them. *Software Engineering* **21**:4 (1995) 314–335
26. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., McDermott, D., Martin, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In: *Proceedings of the first International Semantic Web Conference (ISWC 02)* (2002)
27. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W., Wielinga, B.: *Knowledge Engineering and Management, the CommonKADS Methodology*. MIT Press (2000)
28. Brazier, F.M.T., Jonker, C.M., Treur, J.: Principles of Component-Based Design of Intelligent Agents. *Data and Knowledge Engineering* **41** (2002) 1–28
29. Müller, J.P., Pischel, M.: *The Agent Architecture InteRRaP: Concept and Application*. Technical Report RR-93-26, DFKI Saarbrücken (1993)
30. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI architecture. In: Fikes, R., Sandewall, E. (eds.): *Proceedings of the Second Conference on Knowledge Representation and Reasoning*, Morgan Kaufman, (1991) 473–484
31. Garlan, D., Allen, R., Ockerbloom, J.: Architectural Mismatch, or, Why it's hard to build systems out of existing parts? In: *Proceedings of the Seventh international Conference on Software Engineering*, Seattle, Washington (1995) 179–185
32. van Vliet, H.: *Software Engineering: Principles and Practice*. 1st edn. John Wiley & Sons (1993)
33. de Bruin, H., van Vliet, H.: The Future of Component-Based Development is Generation, not Retrieval. In: Crnkovic, I., Larsson, S., Stafford, J. (eds.): *Proceedings ECBS'02 Workshop on CBSE -- Composing Systems from Components*, Lund, Sweden, April 8–11, (2002)
34. Moormann Zaremski, A., Wing, J.M.: Specification Matching Software Components. *ACM Transactions on Software Engineering and Methodology*, vol. 6:4 (1997) 333–369
35. Brazier, F.M.T., Wijngaards, N.J.E.: Automated (Re-)Design of Software Agents. In: Gero, J.S. (ed.): *Proceedings of the Artificial Intelligence in Design Conference 2002*, Kluwer Academic Publishers (2002) 503–520

36. Schreiber, A. Th., Birmingham, W. P. (eds.): Special Issue on Sisyphus-VT.
37. International Journal of Human-Computer Studies (IJHCS) **44**:3/4 (1996) 275–280
38. Karimi, J., Konsynski, B.R.: An Automated Software Design Assistant. IEEE Transactions on Software Engineering, Vol. 14:2 (1988) 194–210
39. Rich, C., Water. R.C.: The Programmer's Apprentice: A research overview. IEEE Computer **21**:11(1988) 10–25
40. Smith, D.R.: KIDS: A Semi-automatic Program Development System. IEEE Transactions on Software Engineering, Vol. 16:9 (1990) 1024–1043
41. Motta, E., Fensel, D., Gaspari, M., Benjamins, V.: Specifications of Knowledge Component Reuse. In: Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE-99), Germany, Kaiserslautern (1999) 17–19
42. Fensel, D., Motta, E., Benjamins, V., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., van Harmelen, F., Musen, M., Plaza, E., Schreiber, A., Studer, R., Wielinga, B.: The unified problem-solving method development language UPML. Knowledge and Information Systems **5**:1, to appear (2002)
43. Peña-Mora, F., Vadhavkar, S.: Design Rationale and Design Patterns in Reusable Software Design. In: Gero, J., Sudweeks, F. (eds.): Artificial Intelligence in Design (AID'96), Kluwer Academic Publishers, the Netherlands, Dordrecht (1996) 251–268
44. Riel, A.: Object-Oriented Design Heuristics. Addison Wesley Publishing Company, Reading Massachusetts (1996)
45. Kloukinas, C., Issarny, V.: Automating the Composition of Middleware Configurations. In: Proceedings of the 15<sup>th</sup> IEEE International Conference on Automated Software Engineering (2000) 241–244
46. Prieto-Diaz, R., Neighbors, J.M.: Module Interconnection Languages. Journal of Systems and Software **4** (1986) 307–334
47. de Bruin, H., van Vliet, H.: Quality-Driven Software Architecture Composition. Journal of Systems and Software, to appear (2002)
48. de Bruin, H., van Vliet, H.: Top-Down Composition of Software Architectures. In: Proceedings 9th Annual IEEE International Conference on the Engineering of Computer-Based Systems (ECBS), IEEE, April 8-11 (2000) 147–156